УДК 004.27

ПРИМЕНЕНИЕ СОВРЕМЕННЫХ МЕТОДОВ ОБУЧЕНИЯ ДЛЯ РАЗРАБОТКИ МИКРОСЕРВИСОВ НА ПРИМЕРЕ ЯЗЫКА GOLANG

Д. В. Прокопенко

Белорусский государственный университет, пр. Независимости - 4, 220030, Беларусь, diana.prokopenko8888@gmail.com

Рассмотрена проблема применения современных методов обучения при разработке микросервисов на примере языка программирования Golang. Обоснована необходимость использования виртуализации и контейнеризации для обеспечения гибкости и изолированности микросервисов. Результаты данной работы актуальны для преподавателей и студентов, которые заинтересованы в развитии своих знаний и навыков в области разработки микросервисов.

Ключевые слова: язык программирования Golang; обучение студентов; микросервисы; docker; kubernetes.

APPLICATION OF MODERN LEARNING METHODS FOR MICROSERVICE DEVELOPMENT USING GOLANG AS AN EXAMPLE

D. V. Prokopenko

Belarusian State University, Niezavisimosti pr.- 4, 220030, Belarus, diana.prokopenko8888@gmail.com

The problem of applying modern learning methods in the development of microservices is considered using the example of the Golang programming language. The necessity of using virtualization and containerization to ensure the flexibility and isolation of microservices is justified. The results of this work are relevant for teachers and students who are interested in developing their knowledge and skills in the field of microservice development.

Keywords: Golang programming language; student learning; microservices; docker; kubernetes.

Введение

Golang – язык программирования, разработанный компанией Google в 2007 году. Одной из главных особенностей Golang является его эффективность и быстродействие. Важнейшей ценностью для создателей Go была простота, именно поэтому он легок в освоении. Golang идеально подходит для реализации микросервисов. Разработка микросервисов – это практический процесс, и наиболее эффективным методом обучения может быть создание собственных проектов и экспериментирование с ними.

Разработка микросервиса на языке Golang

Рассмотрим разработку сервиса на примере сервиса аутентификации и авторизации.

Аутентификация – это процесс проверки подлинности пользователя. В процессе аутентификации пользователь должен предоставить доказательства своей идентичности такие как логин и пароль. Если эти доказательства верны, то пользователь считается аутентифицированным.

Авторизация – это процесс определения того, имеет ли пользователь право на выполнение определенных действий в приложении. Она основывается на правах доступа, которые были назначены пользователю после его аутентификации. Например, пользователь может иметь права только на чтение определенных данных, но не на их изменение.

Перед тем, как начать разработку данного сервиса, необходимо спроектировать его архитектуру. Для этого необходимо студентов ознакомить с принципами и паттернами проектирования микросервисов, а затем приступить к практической реализации. В проектировании сервиса учитываются следующие аспекты:

1. Безопасность: используются надежные алгоритмы шифрования и хеширования паролей пользователей. Кроме того, важно регулярно обновлять и проверять безопасность сервиса.

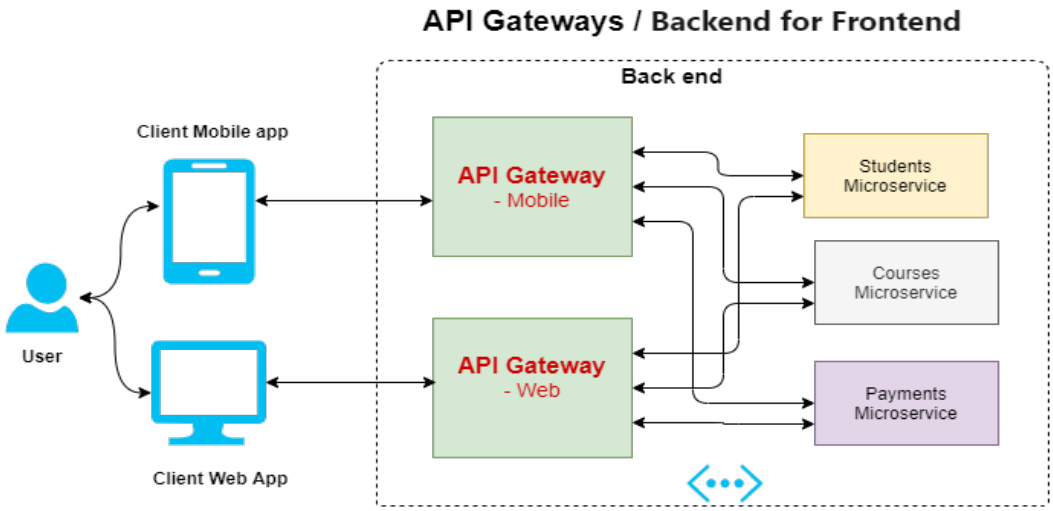
2. Масштабируемость: применяется распределенная архитектура, кеширование и балансировка нагрузки.

3. Гибкость: сервис должен быть гибким и адаптивным к изменяющимся требованиям приложения. Например, если требуется добавить новые методы аутентификации или авторизации, то сервис должен быть легко расширяемым.

4. Переносимость: сервис должен быть переносимым между разными окружениями, такими как разработка, тестирование и продуктивное окружение. Для этого можно использовать контейнеризацию и оркестрацию контейнеров.

Основные паттерны проектирования микросервисов. Стоит отметить, что у каждого паттерна проектирования есть свои недостатки и преимущества. И выбор паттерна зависит от поставленной задачи.

Одним из наиболее распространенных и рекомендуемых паттернов для проектирования сервиса аутентификации и авторизации является паттерн «API Gateway».



Пример API Gateway

API Gateway – предоставляет единую точку входа для клиентских запросов к различным микросервисам. Он выполняет роль маршрутизатора и агрегатора запросов, обеспечивая контроль доступа, аутентификацию и авторизацию.

1. Аутентификация: API Gateway может выполнять проверку подлинности и аутентификацию пользователей, а также генерировать и передавать токены авторизации для доступа к другим микросервисам.

2. Авторизация: API Gateway может выполнять проверку прав доступа пользователей к различным ресурсам и сервисам, используя информацию о пользователях, хранящуюся в сервисе авторизации.

3. Шифрование трафика: API Gateway может использоваться для шифрования и дешифрования трафика между клиентами и микросервисами, обеспечивая безопасность и защиту данных.

4. Кеширование: API Gateway может кэшировать данные, уменьшая нагрузку на микросервисы и ускоряя обработку запросов.

5. Мониторинг: API Gateway может собирать и анализировать данные о запросах и ответах, обеспечивая мониторинг и отладку системы.

Использование паттерна API Gateway может упростить и обезопасить разработку и эксплуатацию сервиса авторизации и аутентификации.

Развертывание микросервисов

Микросервисная архитектура предполагает разбиение приложения на отдельные сервисы, каждый из которых выполняет определенную функцию. Эти сервисы должны быть независимыми друг от друга и могут быть написаны на разных языках программирования. Кроме того, для обеспечения высокой доступности и масштабируемости микросервисов их необходимо разворачивать в различных средах, включая тестовые, продакшн и различные облачные платформы.

Для обеспечения гибкости и изолированности каждого сервиса, часто используют виртуализацию или контейнеризацию. Виртуализация позволяет создавать машины с разными операционными системами, на которых можно запускать различные сервисы. Это позволяет оперативно управлять ресурсами и обеспечивать изолированность каждого сервиса.

Контейнеризация позволяет запускать приложения в контейнерах, которые являются легковесными и могут быть запущены на любой операционной системе, поддерживающей контейнеризацию. Каждый контейнер содержит только необходимые компоненты приложения и его зависимости, что позволяет достичь максимальной гибкости и изолированности.

Студентов следует ознакомить с двумя методами – виртуализацией и контейнеризацией.

Заключение

Развитие микросервисов активно продолжается и охватывает рынок программного обеспечения. Опрос, проведенный среди IT-сотрудников различных стран и должностей, показал, что 49,3% считают микросервисную архитектуру стандартом для больших и сложных проектов, а 36,2% уверены, что она станет промышленным стандартом для разработки бэкенда.

Таким образом, разработка микросервисов на Golang с использованием современных методов и технологий имеет большой потенциал для создания безопасного и эффективного сервиса.

Библиографические ссылки

1. API Gateway [Электронный ресурс]. URL: https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/ (дата обращения: 10.08.2021).

2. Golang [Электронный ресурс]. URL: https://go.dev/

3. State of Microservices 2020 [Электронный ресурс] – URL: https://tsh.io/state-of-microservices/#future (дата обращения: 01.05.2021).